

Guide - Git & Gitlab

ING2 & M1

Editeur :

Victor Coindet

ENSG

6 et 8 Avenue Blaise Pascal

Cité Descartes

77420 Champs-sur-Marne

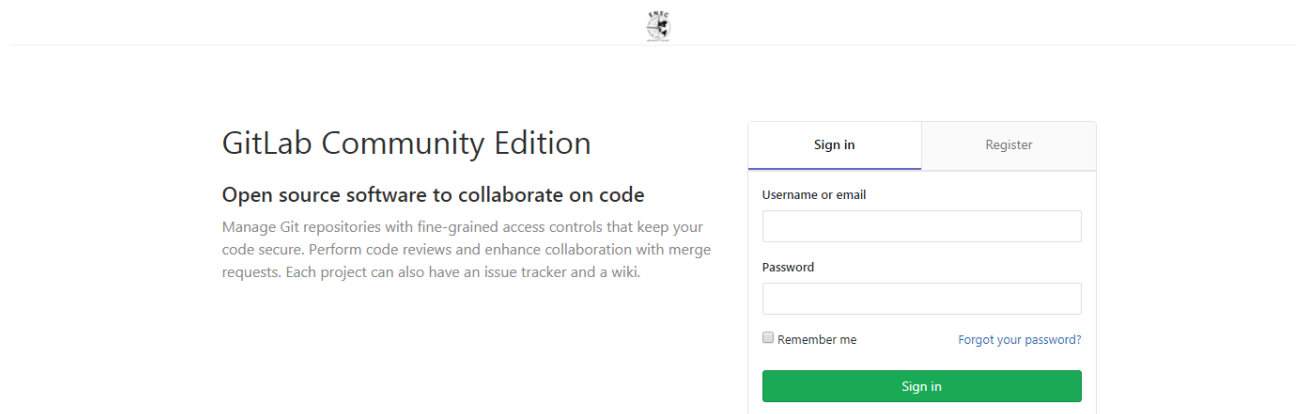
France

Tel : +33 (0) 1 64 15 31 21

E-mail : <victor.coindet@ensg.eu>

1 Se connecter

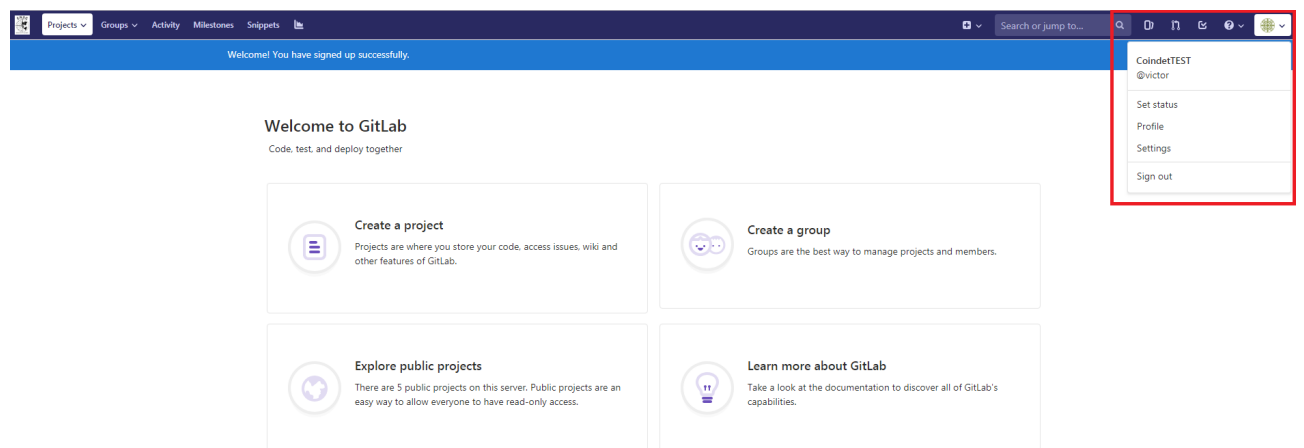
Aller sur <http://gitlab-ing2.ensg.eu> *Attention* : Ce site n'est accessible qu'en intranet, de plus il ne sera disponible que le temps de la formation. Un autre gitlab sera monté (en interne encore) pour les projets développements.



Créez vous un compte en cliquant sur "Register" ou connectez vous en cliquant sur "Sign In" si vous avez déjà un compte.

Remplissez tous les champs et soumettez les. Rentrez une adresse mail valide ! ;)

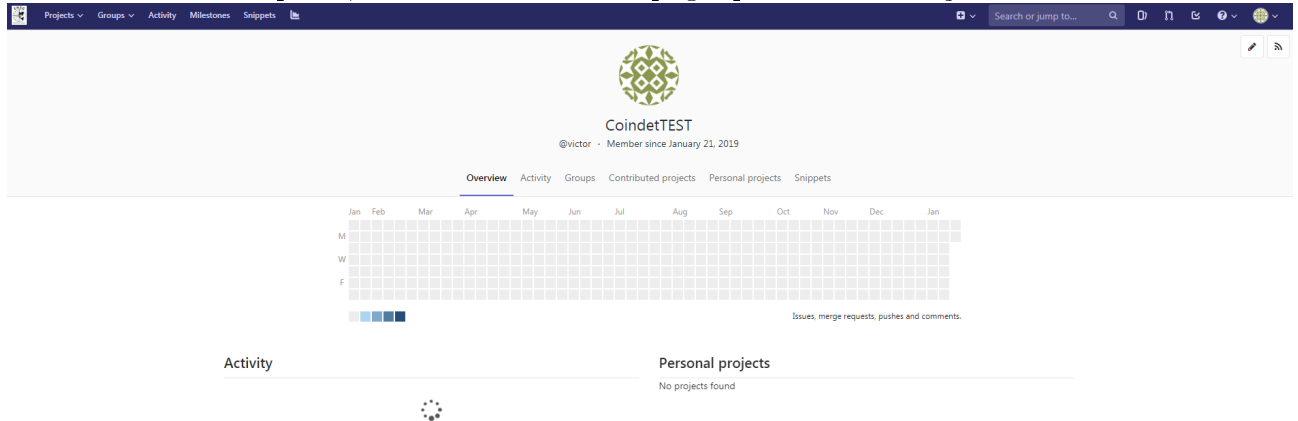
Une fois connecté, vous devez arriver sur la page suivante :



2 Consulter son profil

En cliquant sur l'icône en haut à droite (voir encadré rouge sur figure précédente), il vous est possible d'accéder à votre profil et de configurer votre compte gitlab ("Settings" : modification du nom, changement de mot de passe, changement de l'adresse mail, ajout d'une clé SSH, etc.).

En allant sur votre profil, vous arrivez sur une page qui ressemble à ça :



Vous retrouverez ici les répertoire ou les dépôts git que vous avez cloné sur votre propre compte gitlab. Comme vous venez de créer votre compte, vous n'avez aucun projet en cours. Pour avoir des projet, il faut faire un fork de projets déjà existant, ou en créer vous même. Vous pourrez aussi retrouver sur cette pages vos contribution aux différents projets auxquels vous participez.

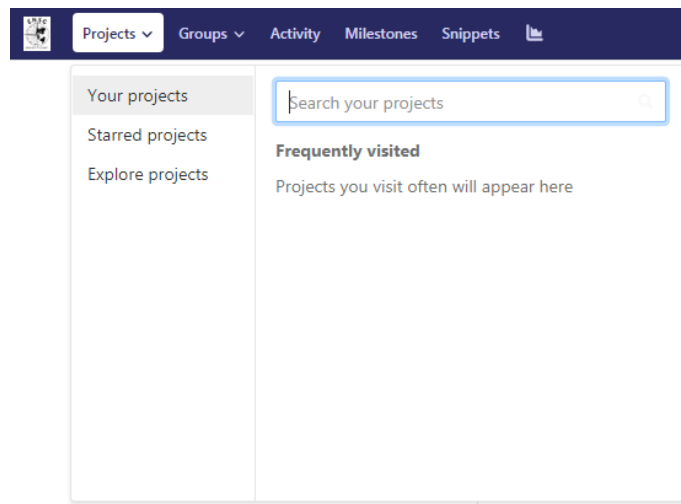
3 Créer un nouveau répertoire Gitlab - Scrum Master

Il existe plusieurs façon de créer un répertoire Gitlab. Ce tutoriel vous propose d'en voir deux :

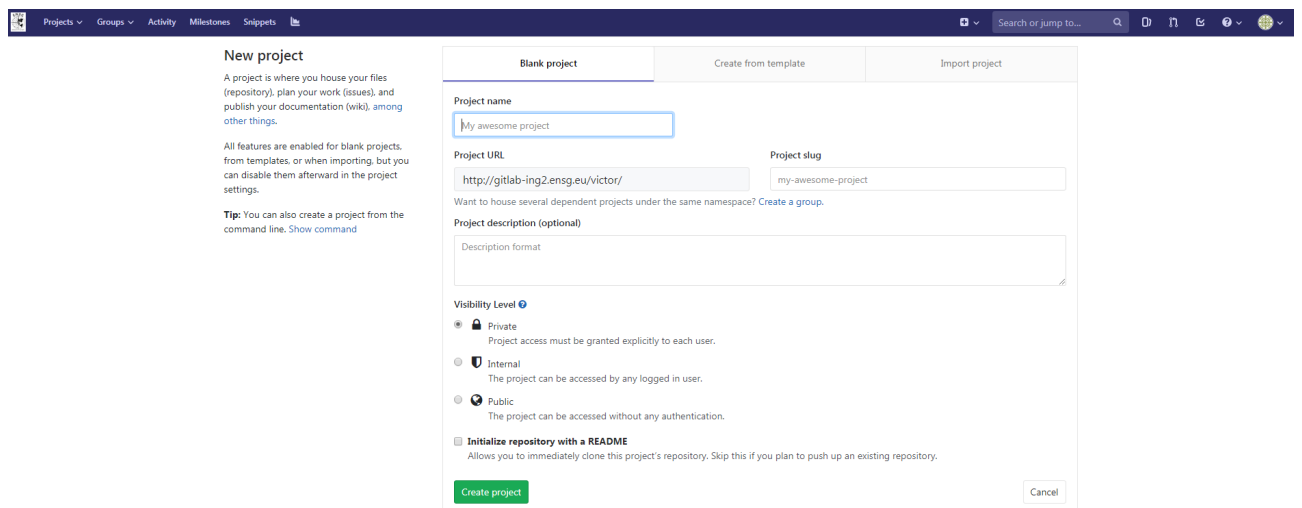
- From scratch
- Depuis un répertoire déjà existant : le Fork !

3.1 From Scratch

Pour créer un nouveau projet, sélectionnez l'onglet 'Projects' en haut à gauche, puis 'Your projects'.

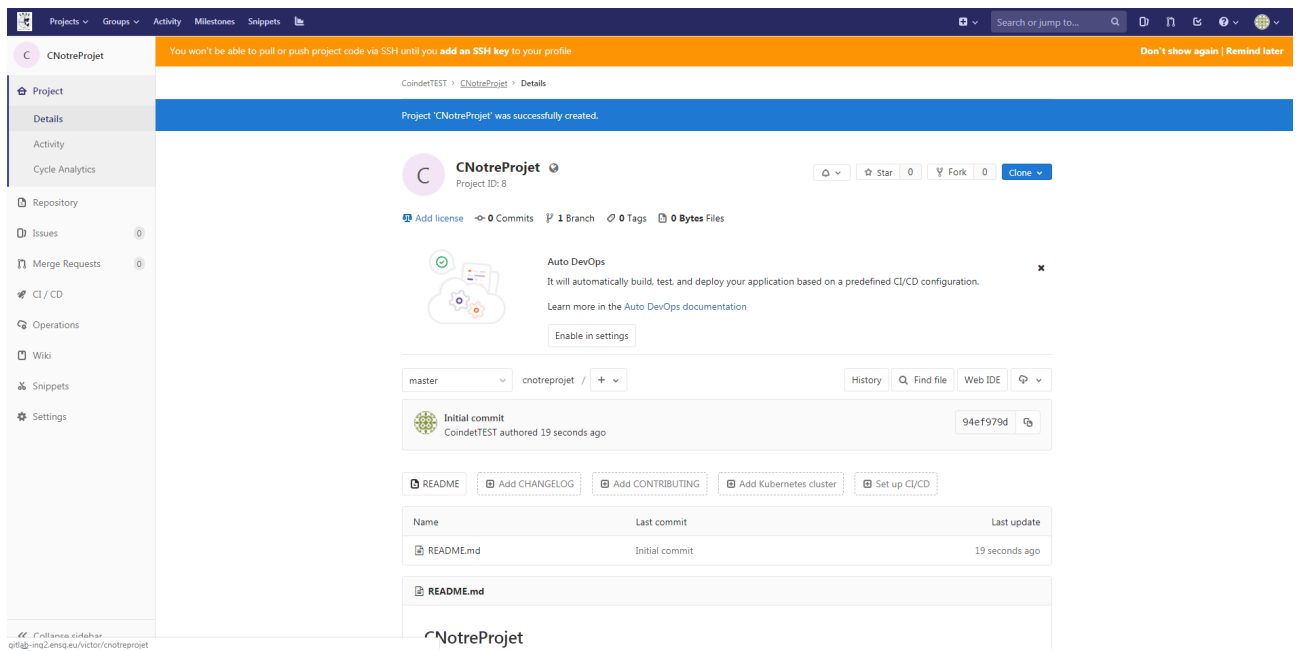


Cliquez ensuite sur le cadre "Create a project". Vous devez arriver sur la page suivante :



Renseignez le nom du projet, et mettez le 'Public' (pour nous puissions y avoir accès, nous les profs). **Initialisez le avec un README**

Vous devez maintenant avoir une page qui ressemble à celle ci :

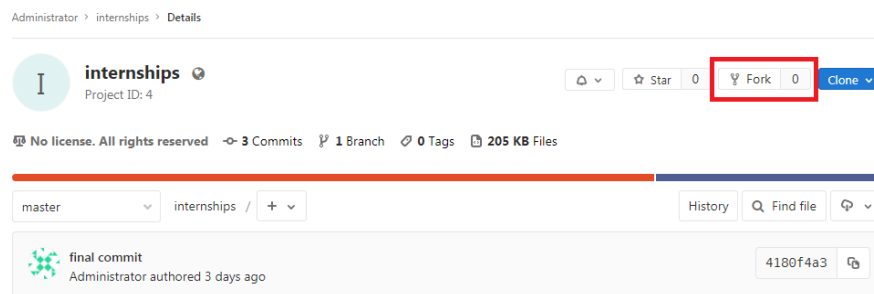


3.2 Depuis un répertoire déjà existant : le Fork !

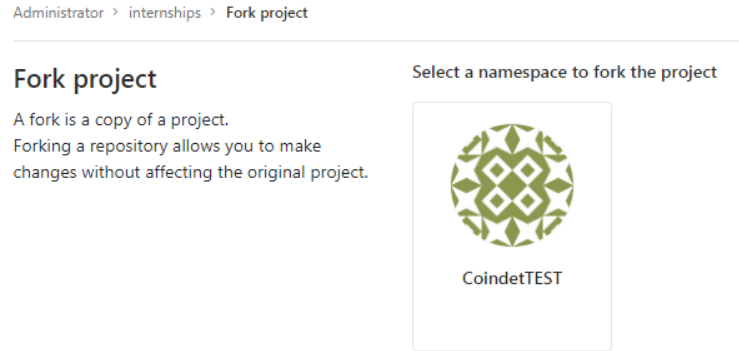
Vous trouverez les dépôts des projets existant ici : <http://gitlab-ing2.ensg.eu/explore/projects>.
Ce sont les projets mis à disposition pour le cours

L'idée du 'Fork' est de copier un projet déjà existant sur votre propre compte.

Sélectionnez le projet que vous voulez copier, puis cliquez sur le bouton 'fork' :



Vous devez arriver sur la page suivante qui vous demande le 'namespace' où copier le projet.
Vous ne devez avoir qu'un seul namespace : votre compte !

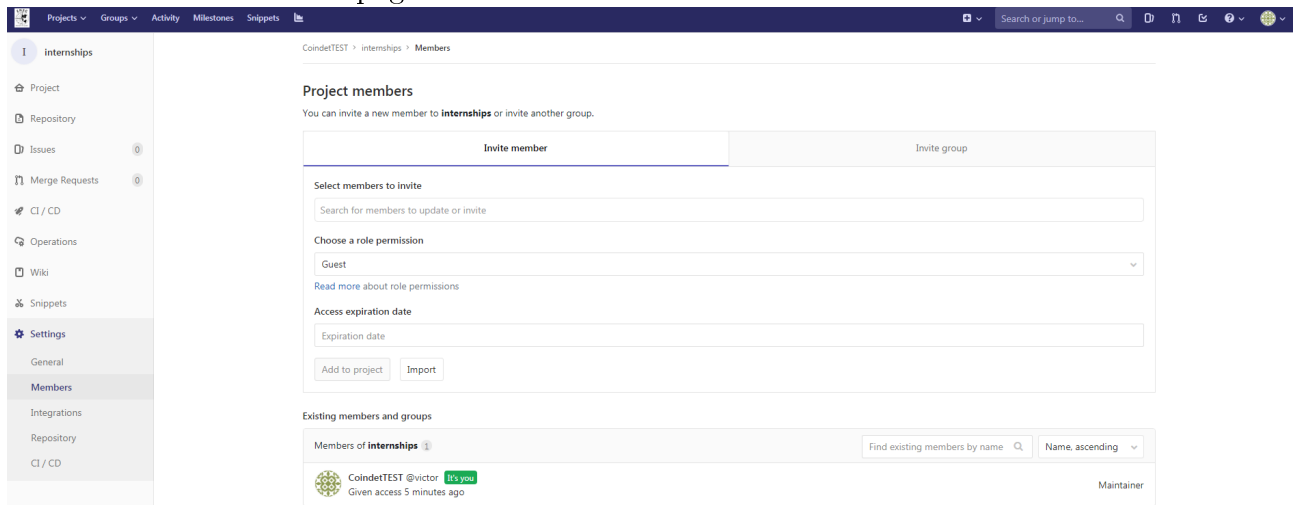


Une fois le projet forké, vous devez le retrouver dans vos propres projets ('Projects' -> 'Your projects').

4 Ajouter des développeurs aux projets

Pour ajouter des développeurs à votre projet, il vous suffit d'aller dans le menu 'Settings' -> 'Members' (en bas à gauche une fois que vous avez sélectionné votre projet).

Vous devez arriver sur la page suivante :



Vous pouvez sélectionner un utilisateur via son login gitlab, ou son adresse mail.

Dans la partie 'Choose a role permission', donnez lui la le rôle 'Developer', puis cliquez sur 'Add to project'. Il sera toujours possible de changer ses droits en revenant sur cette page.

5 Protéger la branche Master

Sur Gitlab, la branche master est protégée par défaut. C'est-à-dire que seul le propriétaire(ou 'Maintener') du projet peut pousser des modifications sur cette branche. Les développeurs ne pourront pas pousser leur modifications sur cette branche, sans l'accord du Maintener, c'est-à-dire que les développeurs devront faire des merge request (ou pull request).

Cette configuration peut être changée dans 'Settings' -> 'Repository' -> 'Protected Branches'. Il est cependant déconseiller de modifier cela.

6 Git- Configurer le proxy

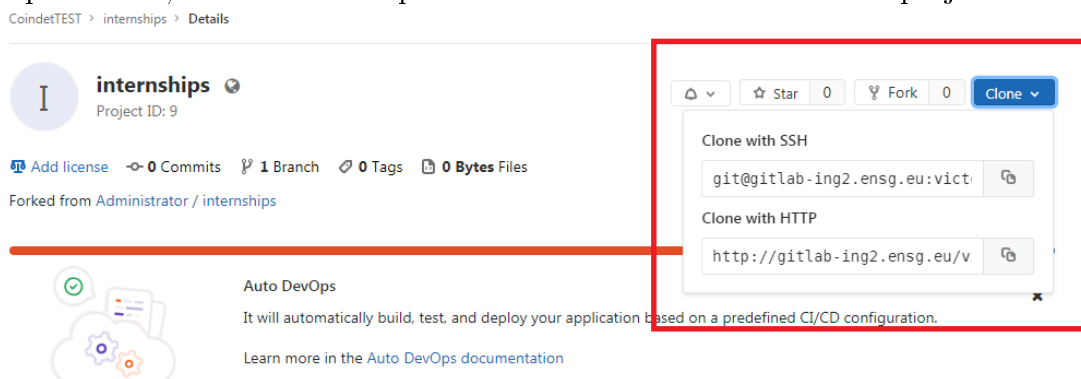
Il n'est normalement pas nécessaire de configurer le proxy de Git pour cette formation, mais dans le doute... Pour modifier le proxy, vous devez depuis un terminal, taper la commande suivante :

```
git config --global http.proxy http ://10.0.4.2 :3128
```

10.0.4.2 :3128 étant le proxy de l'école.

7 Cloner un dépôt - Développeurs

Commencez par demander à votre scrum master l'adresse du fork qu'il a fait sur son propre compte Gitlab, ce sera votre dépôt de référence sur tout le reste du projet.



Lorsque vous cliquez sur "Clone", gitlab vous donne l'adresse du répertoire soit sous le protocole HTTP, soit sous le protocol SSH (ici `http ://gitlab-ing2.ensg.eu/victor/internships.git`). Copiez l'adresse sous le protocol HTTP.

Dans un terminal, placez vous à l'endroit où vous souhaitez cloner le dépôt. Dans votre terminal tapez la commande suivante :

```
git clone http ://gitlab-ing2.ensg.eu/victor/internships.git
```

Pensez bien a mettre la bonne adresse, et non celle qui est écrite ci-dessus.

Attention : *Nous ne travaillons que très rarement sur la branche Master, il va donc falloir créer une nouvelle branche en local sur votre propre machine pour y faire vos modifications.*

8 Créer une branche

Dans votre terminal, placez-vous dans le répertoire que vous venez de cloner. Pour créer une nouvelle branche, il suffit de lancer la commande :

git checkout -b name_of_your_new_branch

checkout : permet de changer de branche.

-b : permet de créer la nouvelle branche.

Cette commande crée donc une nouvelle branche et vous place directement sur celle-ci.

Lorsque vous clonez un projet, vous clonez toutes les branches qui vont avec, pas seulement la branche master.

9 Changer de branche

Si vous souhaitez vous placer sur une branche existante en local, il vous suffit de lancer la commande :

git checkout name_of_your_branch

VOUS POUVEZ MAINTENANT COMMENCER A CODER !;)

10 Faire un commit

Un commit est un message avant validation des modifications. Ces messages permettent de décrire les modifications apportées au code, mais aussi d'avoir un historique de commits qui permet aux utilisateurs de se reporter rapidement aux modifications antérieurs. Exemple d'historique :

Message	Date	Author
◉ master ◀13 fix(core): marked dependency is insecure version	08/01/2018 13:45	Gerald Choqueux
◉ feat(3dtiles): add wireframe support for 3dTiles layer, and add wireframe checkbox to 3...	21/12/2017 15:19	Guillaume Liegard
◉ docs(jsdoc) : adding a home page to the API documentation + adding a new template...	20/12/2017 16:57	vcoindet
◉ feat(wfs): Add a debug UI for geometry layer, to change visibility, opacity and toggle w...	20/12/2017 15:26	Guillaume Liegard
◉ fix(wfs): use bigger integer data type for indices array, for THREE.LineSegments. J Prior...	20/12/2017 10:53	Guillaume Liegard
◉ fix(protocols): stop parsing unnecessarily xbil buffer	18/12/2017 14:38	Gerald Choqueux
◉ examples(wfs): Place points and bus lines on the ground	12/12/2017 16:51	Guillaume Liegard
◉ refactor(wfs): add contour coordinates to the callback that compute altitude.	12/12/2017 16:42	Guillaume Liegard
◉ fix (renderer): set DEBUG after defines definition	18/12/2017 13:35	Adrien Berthet
◉ 2.2.0	15/12/2017 20:21	Augustin Trancart
◉ fix (protocols) : stop using layer's option tileMatrixSet in WMS provider	24/10/2017 13:56	Gerald Choqueux
◉ fix (examples): Wrong zoom min in layer example	13/12/2017 15:11	Gerald Choqueux
◉ example: complete the index one with five different layers	12/12/2017 10:43	Adrien Berthet
◉ fix (example): set correct value on init of GUITools	12/12/2017 09:46	Adrien Berthet
◉ feat (core): complete and use FrameRequesters	05/12/2017 15:30	Adrien Berthet
◉ docs (core): fix error to build view documentation	05/12/2017 15:03	Gerald Choqueux
◉ chore: add out/ to .gitignore	05/12/2017 10:35	Adrien Berthet
◉ fix(pointcloud): don't ignore layer.opacity	15/11/2017 11:05	Augustin Trancart
◉ chore(example): enlarge debug ui in pointcloud.html	15/11/2017 15:56	Augustin Trancart
◉ fix(pointcloud): add internal groups before first update	07/11/2017 14:10	Augustin Trancart
◉ fix(pointcloud): honour object3D for pointcloud layers	06/11/2017 16:36	Augustin Trancart
◉ fix (examples): add comments to panorama.html and remove arrow func	21/11/2017 10:51	pierre-eric
◉ fix (core): add quality settings to tweak panorama refinement behavior	21/11/2017 10:51	pierre-eric
◉ fix (geometry): allow layer to override the default 16 segments value	21/11/2017 10:49	pierre-eric

Sur cet exemple, il est possible de se resituer sur n'importe quel commit, et de ne plus avoir les modifications apportées au code après ce commit. Ce genre de situation peut arriver pour retrouver à quel moment un bug a pu s'insérer et le corriger plus rapidement.

AVANT DE COMMITER

Avant de commiter, pour voir quels fichiers ont été modifiés, vous pouvez lancer la commande :

git status

S'il y a eu des modifications, git vous le signal en affichant le message suivant :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsp\iTownspProject (myBranch2)
λ git status
On branch myBranch2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   test.js

no changes added to commit (use "git add" and/or "git commit -a")
```

Ici, Git nous signale que le fichier test.js a été modifié. Il faut donc l'indexer afin de préparer le prochain commit. La commande à lancer est :

git add nom_du_fichier_modifié

ou

git add .

Cette dernière commande permet d'indexer toutes les modifications sur tous les fichiers d'un coup.

Les modifications sont désormais notées comme validées et indexées. Vous pouvez maintenant commiter :

git commit --m "message du commit"

Notez qu'il est important de laisser un message car c'est ce message que les autres développeurs vont lire, il donne une indication sur vos modifications.

A présent, si vous faire un nouveau git status, vous devez avoir :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (myBranch2)
λ git status
On branch myBranch2
nothing to commit, working tree clean
```

11 Tirer les sources - Pull

Pendant que vous étiez en train de faire vos propres modifications, il est possible que d'autres développeurs aient apporté des modifications, qu'ils les aient déjà poussées sur le serveur Github et que ces modifications soient déjà fusionnées avec la branche master. Votre branche master locale et la branche master du serveur ne sont donc plus identiques... Imaginez que vous poussiez vos modifications et qu'elles soient fusionnées avec la branche master : toutes les modifications de vos collègues seront écrasées, ce qui est dommage. Pour éviter ce genre de problème, il faut d'abord tirer les sources sur votre propre machine afin de mettre à jour votre branche master à jour. Afin de prendre en compte les modifications qui ont été poussées sur le serveur distant par d'autres développeurs, il est nécessaire de les rapatrier sur sa propre machine. On fait ce qu'on appelle un git pull :

git pull

12 Faire un "rebase"

Dans les bonnes pratiques, avant de pousser ses modifications sur le dépôt Github, il est bien de mettre sa branche à jour avec la branche master. En effet, il se peut que des personnes aient déjà poussé des modifications avant vous, modifications que vous n'avez pas forcément encore pris en compte sur votre machine en local.

Il faut donc faire ce qu'on appelle un « rebase », les étapes à suivre sont :

git add .

```
> Cmdr
pick 8e24f5e test
pick e541e92 test2
pick 62896c4 test

# Rebase c2da0ff..62896c4 onto c2da0ff (3 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
~
~
~ # This commit was squashed!
~
~ # This commit was squashed!
~
~ # This commit was squashed!
~
~ # This commit was squashed!
~
~ # This commit was squashed!
~
~
~
> GestionProjetInformatique/Projets/iTownsP/iTownsProject/.git/rebase-merge/git-rebase-todo [unix] (17:15 17/01/2018),1,1 All
<GestionProjetInformatique/Projets/iTownsP/iTownsProject/.git/rebase-merge/git-rebase-todo" [unix] 22L, 694C
```

- **pick** : on prend en compte le commit tel quel
- **reword** : on souhaite modifier le message du commit
- **edit** : on souhaite modifier complètement un commit (pour le couper en deux par exemple)
- **squash** : le commit se fusionné avec le commit précédent
- **fixup** : comme squash, mais utilise le message du commit précédent
- **exec** : lorsqu'on souhaite ajouter une ligne de commande
- **drop** : on supprime un commit.

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (myBranch)
λ git rebase -i origin/master
Successfully rebased and updated refs/heads/myBranch.
```

11

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (myBranch)
λ git rebase -i origin/master
Auto-merging index.html
CONFLICT (add/add): Merge conflict in index.html
error: could not apply 8e24f5e... test

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".

Could not apply 8e24f5e... test
```

Git a repéré des conflits entre votre code et celui de la branche master. Il faut donc régler ces conflits à l'aide d'un éditeur de texte. Là où il y a des conflits, git a inséré des lignes comme les suivantes :

```
64 <<<<<< HEAD
65 =====
66 |         |         |         | lalalalalala
67 >>>>>> 8e24f5e... test
```

« HEAD » + le numéro du commit.

C'est à vous de juger ce qui doit être pris en compte pour régler le conflit. Une fois les conflits réglés, pensez bien à enregistrer vos fichiers, puis lancez les commandes suivantes :

- **git add .**
- **git rebase --continue**

Une fois que le rebase est terminé, vous devez avoir ce message :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (HEAD detached at c2da0ff)
λ git rebase --continue
Successfully rebased and updated refs/heads/myBranch.
```

Vous êtes maintenant prêt pour pousser vos modifications sur le serveur distant. Il est toujours possible d'abandonner un rebase en lançant la commande :

git rebase --abort

C'est un retour à la case départ, comme si vous n'aviez jamais lancé de rebase.

13 Push

Pour pousser votre code (ou votre branche), il suffit de lancer la commande :

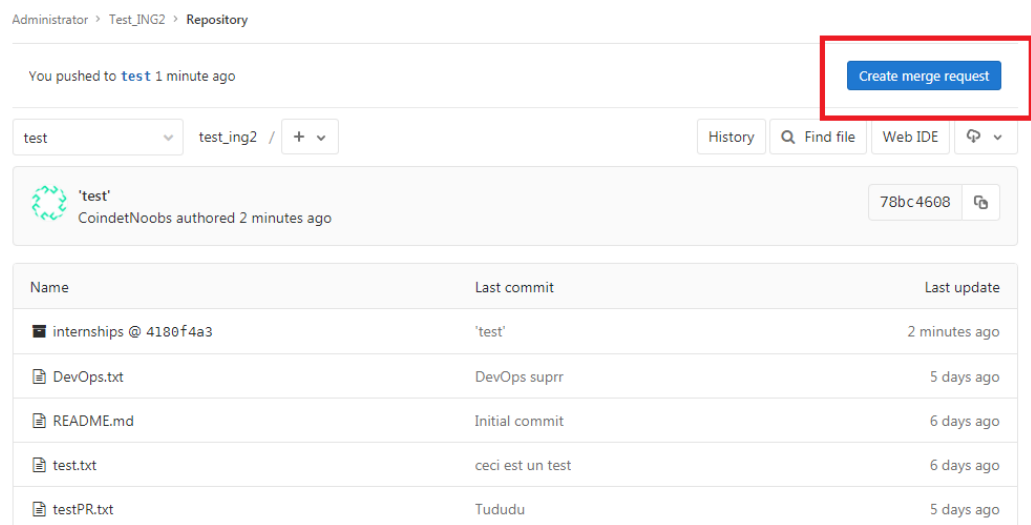
git push

Attention, pensez bien à commiter et faire vos rebases avant de pousser votre code.

14 Faire une Merge Request sur Gitlab

Une Merge Request est une demande de fusion de sa branche avec une autre branche (souvent la master). Une fois qu'une branche est poussée sur le serveur il est possible de réaliser une telle demande sur gitlab directement :

Lorsque nous sommes dans le projet sur gitlab, il suffit de se placer sur la branche que vous voulez merger. A ce moment, un bouton "Create merge request" en haut à droite.



Une fois que vous avez cliqué sur ce bouton, vous arrivez sur une page qui propose de merger votre branche sur la branche master 'From branch into master', vous pouvez bien entendu changer les branches que vous voulez fusionner.

Pensez bien à mettre un nom à votre merge request, ainsi qu'une description (indiquer les modifications réalisées, les fichiers modifiés, dans quel cadre ce développement a été réalisé, etc.)

15 Accepter uen Merge Request (Scrum Master)

Dans l'onglet "Merge Requests" à gauche, le scrum master trouvera les différentes demandes de merge request.

En cliquant sur une merge request, le scrum master peut l'accpter en cliquant sur 'Merge' ou la refuser en cliquant 'Close merge request'.

Une fois la merge request acceptée, gitlab se charge de fusionner (merger) les deux branches, c'est-à-dire insérer les modifications de notre branche de développement dans la branche master.

Lorsqu'un merge a été fait sur la branche master, il est de bonne rigueur d'effectuer un rebase sur sa propre branche de développement pour être sûr d'être bien à jour avec le code sur le serveur.